НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування і спеціалізованих комп'ютерних систем

«До захисту допущено»

Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

"___" червня 2019 р.

Дипломна робота

на здобуття ступеня бакалавра

зі спеціальності **6.050102 «Комп'ютерна інженерія»**

на тему: АЛГОРИТМ ДИФЕРЕНЦІАЛЬНОЇ ЕВОЛЮЦІЇ ДЛЯ ГЛОБАЛЬНОЇ_____ ОПТИМІЗАЦІЇ _____
_____

Виконав (-ла): студент (-ка) IV курсу, групи __КВ-53___
(шифр групи)

Озеров Микола Сергійович _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Керівник доц.каф.СКС, к.т.н., доц., Зорін Ю. М. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц. каф. СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) _____
(підпис)

Рецензент професор каф. ОТ, д.т.н., проф. Кулаков Ю. О. ___ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____ (підпис)

Київ – 2019 року

Національний технічний університет України

“Київський політехнічний інститут” імені Ігоря Сікорського

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Освітньо-кваліфікаційний рівень “Бакалавр”

Напрям підготовки 6.050102 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ В.П. Тарасенко

“___” _____

2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Озерову Миколі Сергійовичу

1. Тема роботи: АЛГОРИТМ ДИФЕРЕНЦІАЛЬНОЇ ЕВОЛЮЦІЇ ДЛЯ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ,
керівник роботи: Зорін Юрій Михайлович, к.т.н., доцент,
затверджені наказом по університету від “__” _____ 2019 року № ____.

2. Строк подання студентом роботи: “__” червня 2019 р.

3. Вихідні дані для дипломної роботи:
- модифікована версія алгоритму диференціальної еволюції;
- результати тестів базової версії алгоритму відносно інших відомих методів глобальної оптимізації;
- результати тестів модифікованої версії алгоритму відносно базової.

4. Перелік задач, які потрібно вирішити:

– вивчити літературні джерела за тематикою дослідження, в тому числі методики глобальної оптимізації;
– провести аналіз алгоритмів, які використовуються для глобальної оптимізації;
– обґрунтувати вибір методу глобальної оптимізації;
– розробити комп'ютерну модель алгоритму;
– оцінити переваги та недоліки алгоритму.

5. Перелік обов'язкового ілюстративного матеріалу:
   – схема алгоритму диференціальної еволюції;
   – схема мутацій алгоритму диференціальної еволюції;
   – псевдокод модифікованого алгоритму диференціальної еволюції;
   – таблиця результатів.

6. Дата видачі завдання: "__" _____ 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строк виконання етапів	Примітка
1.	Вивчення літератури за тематикою дипломної роботи	15.01.2019	
2.	Розроблення та узгодження завдання	24.01.2019	
3.	Підготовка матеріалів першого розділу дипломної роботи	10.02.2019	
4.	Підготовка матеріалів другого розділу дипломної роботи	25.03.2019	
5.	Розробка модифікованої версії алгоритму диференціальної еволюції	10.04.2019	
6.	Підготовка матеріалів третього розділу дипломної роботи	14.04.2019	
7.	Підготовка матеріалів четвертого розділу дипломної роботи	28.04.2019	
8.	Підготовка графічної частини дипломної роботи	20.05.2019	
9.	Оформлення дипломної роботи	30.05.2019	

Студент _____ Озеров М. С.

Керівник роботи _____ Зорін Ю. М.

АНОТАЦІЯ

Об’єкт розробки – процес глобальної оптимізації для розв’язання задачі пошуку глобальних екстремумів функції багатьох змінних.

Мета роботи – розробка модифікованого алгоритму диференціальної еволюції, який має більш якісні характеристики с точки зору точності й сталості результатів.

У роботі розкрито можливості алгоритму диференціальної еволюції. Було досліджено переваги та недоліки відносно інших метаевристичних алгоритмів.

Розроблена модифікація для алгоритму диференціальної еволюції, яка помітним чином покращує процес пошуку глобального екстремуму.

Тестування проводились на відомих функціях оцінки метаевристичних алгоритмів. Знайдені оптимальні параметри алгоритму. Робота була виконана на мові програмування C++.

Ключові слова:

ДИФЕРЕНЦІАЛЬНА ЕВОЛЮЦІЯ, ГЛОБАЛЬНА ОПТИМІЗАЦІЯ, МЕТАЕВРИСТИЧНІ АЛГОРИТМИ, ФУНКЦІЇ БАГАТЬОХ ЗМІННИХ, МОДИФІКОВАНІ АЛГОРИТМИ.

АННОТАЦИЯ

Объект разработки - процесс глобальной оптимизации для решения задачи поиска глобальных экстремумов функции многих переменных.

Цель работы - разработка модифицированного алгоритма дифференциальной эволюции, который имеет более качественные характеристики с точки зрения точности и устойчивости результатов.

В работе раскрыты возможности алгоритма дифференциальной эволюции. Было исследовано преимущества и недостатки относительно других метаэвристических алгоритмов.

Разработана модификация для алгоритма дифференциальной эволюции, заметным образом улучшает процесс поиска глобального экстремума.

Тестирование проводились на известных функциях оценки метаэвристических алгоритмов. Найдены оптимальные параметры алгоритма. Работа была выполнена на языке программирования C ++.

Ключевые слова:

ДИФФЕРЕНЦИАЛЬНАЯ ЭВОЛЮЦИЯ, ГЛОБАЛЬНАЯ ОПТИМИЗАЦИЯ, МЕТАЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ, ФУНКЦИИ МНОГИХ ПЕРЕМЕННЫХ, МОДИФИЦИРОВАНЫЕ АЛГОРИТМЫ.

ABSTRACT

The object of development is the global optimization process for solving the problem of searching for global extremums of the function of many variables.

The purpose of the work is to develop a modified algorithm for differential evolution, which has more qualitative characteristics in terms of accuracy and sustainability of the results.

The paper describes the possibilities of the algorithm for differential evolution. The advantages and disadvantages of other meta-heuristic algorithms were explored.

A modification was developed for the differential evolution algorithm, which greatly improves the process of finding a global extremum.

Testing was carried out on well-known functions of estimating meta-heuristic algorithms. The optimal parameters of the algorithm are found. The work was done in the programming language C ++.

Keywords:

DIFFERENTIAL EVOLUTION, GLOBAL OPTIMIZATION, METAHEURISTIC ALGORITHMS, FUNCTIONS OF MANY VARIABLES, MODIFIED

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 3

ВСТУП 4

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОЇ РОБОТИ 6

1.1. Об’єкт дослідження та постановка задачі 6

1.2. Приклади застосування глобальної оптимізації 7

1.3. Відомі підходи до розв’язання задачі 9

2. МЕТАЕВРИСТИЧНІ АЛГОРИТМИ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ 19

2.1. Паралельні генетичні алгоритми(PGA) 19

2.2. Алгоритм Grefensstett’a 21

2.3. Алгоритм Eshelman’a 22

3. РОЗРОБКА АЛГОРИТМУ ДИФЕРЕНЦІАЛЬНОЇ ЕВОЛЮЦІЇ ДЛЯ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ 24

3.1. Передумова 24

3.2. Базовий алгоритм 25

3.3. Візуалізація роботи базового алгоритму 28

3.4. Модифікований алгоритм диференціальної еволюції 38

4. ТЕСТУВАННЯ АЛГОРИТМУ 40

4.1. Функції використані при тестуванні 40

4.2. Дослідження впливу налаштувань алгоритму 44

4.3. Порівняння базової версії алгоритму відносно інших метаевристичних методів 44

4.4. Порівняння базової та модифікованої версії алгоритму 45

4.5. Аналіз результатів 48

ВИСНОВКИ 49

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 50

ДОДАТКИ 51

Додаток 1. Копії графічних матеріалів 52

Схема алгоритму DE 52

Схема мутацій алгоритму DE 53

Псевдокод модифікованого алгоритму DE 54

Таблиця результатів 55

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

Differential evolution (DE) – диференціальна еволюція.

Genetic algorithm (GA) – генетичний алгоритм.

Evolutionary algorithm (EA) – еволюційний алгоритм.

Parallel genetic algorithm (PGA) – паралельний генетичний алгоритм.

Branch and Bound (BnB) – метод гілок і меж.

Nondeterministic polynomial (NP) – клас задач, точна відповідь на які не може бути визначена за поліноміальний час.

Modified differential evolution (MDE) – модифікована версія алгоритму диференціальної еволюції.

Multiple instruction, multiple data (MIMD) – схема роботи, при якій існує декілька обчислювальний пристроїв, які виконують свій набір команд.

ВСТУП

У наш час швидкого технологічного прогресу, задачі оптимізації набирають все більший пріоритет для їх вирішення. Багато нових теоретичних, алгоритмічних і обчислювальних внесків оптимізації були використані для вирішення проблем в галузі науки і техніки.

Концепція оптимізації є нестаріючою, і є вбудована в природу нашого всесвіту. Вона починається в мікрокосмі, де атоми намагаються сформувати зв'язки, щоб мінімізувати енергію їх електронів. Це також можна побачити в дарвінівському принципі виживання найбільш пристосованих, який разом з біологічною еволюцією призводить до кращої адаптації видів до їхнього оточення. Тут місцевий оптимум є добре адаптованим видом, який домінує над усіма іншими тваринами в його оточенні. Оптимізація також відіграє значну роль у нашому повсякденному житті. Ми прагнемо досягти найкращого за певних обставин. Прикладами є пошук найкоротшого маршруту, оптимального бюджетування, максимізація прибутку та продажів, зберігаючи при цьому найнижчу вартість і т. д.

Для задачі глобальної оптимізації складних функцій багатьох змінних основною проблемою є вибір потрібної евристики. Якщо певний алгоритм не може бути визначеним для розв’язку конкретної задачі з таких причин, як наявність декількох локальних екстремумів або ж не диференційованість, то можна скористатись еволюційними алгоритмами, які мають потенціал обійти ці обмеження [1].

Еволюційні алгоритми є алгоритмами прямого пошуку. Такі алгоритми використовують стратегію генерації параметрів векторів. Як тільки змінна генерується, новий векторний параметр приймається, якщо він зменшує значення цільової функції. Такий метод зазвичай називають жадібним пошуком. Жадібний пошук має швидке сходження, але може потрапити в локальні екстремуми, що майже анігілює знаходження глобального. Цей недолік може бути виправленим, якщо використовувати кілька векторів рішень.

Саме таку ідею використовують еволюційні алгоритми. Найбільш популярним еволюційним алгоритмом є генетичний алгоритм. На даний момент часу існує велика кількість генетичних алгоритмів, але всі вони вимагають значного часу виконання. Для того, щоб подолати цей недолік, еволюційна стратегія, названа диференціальною еволюцією, була запропонована Райнером Сторном в 1997 році [2].

У даній роботі було розроблено модифіковану версію алгоритму диференціальної еволюції для вирішення задачі глобальної оптимізації функції багатьох змінних, яка помітно випереджає звичайну версію алгоритму.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОЇ РОБОТИ

1.1. Об’єкт дослідження та постановка задачі

Задачу глобальної оптимізації можна представити в такому вигляді:

$$\min f(x) \text{ за умови, що}$$
$$x \in D, \text{ де } D = \{x : l \leq x \leq u;\ g_j(x) \leq 0\ j = 1, \dots, j\}$$

- $x \in \mathbb{R}^n$: дійсний n-вимірний вектор змінних.
- $f : \mathbb{R}^n \to \mathbb{R}$: деяка функція.
- $D \subset \mathbb{R}^n$: непорожній набір прийнятих рішень.

D є визначеною такими параметрами:

- *l* та *u*: верхня та нижня границя пошуку для певної змінної.

Опуклість цільової функції не гарантується, також вона може мати багато локальних екстремумів, може мати розриви, тому математично чіткого формулювання не існує.

Дана задача є NP-складною (точне рішення не може бути знайдене за поліноміальний час). Більшість задач даного класу розв’язуються за допомогою методів наближених рішень таких, як евристичні алгоритми.

Основна різниця між легкими та складними проблемами полягає в опуклості або ж не опуклості функції оптимізації.

Для опуклої функції локальний екстремум буде також і глобальним екстремумом, що дуже сильно спрощує задачу його знаходження.

У неопуклої функції може існувати багато локальних екстремумів, що сильно ускладнює задачу оптимізації і навіть робить неможливою для деяких алгоритмів.

У такому випадку можна використати евристичні методи глобальної оптимізації, такі як еволюційні алгоритми.

1.2. Приклади застосування глобальної оптимізації

Передбачення структури білків – є виведенням трьох-вимірної структури білків з послідовності амінокислот. Дана проблема є однією із найбільш важливіших цілей біоінформатики та теоретичної хімії. Протеїни – це ланцюги амінокислот з'єднаних разом пептидними зв'язками. Майже всі протеїни є набором із 20 амінокислот (є рідкі виключення і модифікації), але що робить їх різними – їх форма. Протеїн може навіть не буде визначеним протеїном, якщо він розташований у неправильній формі.

У неправильній формі, в найкращому випадку, протеїн не буде виконувати свою функцію. В найгіршому – може вбити людину.

Задача комівояжера – представляє великий клас задач відомих, як задачі комбінаторної оптимізації. Зазвичай дається певна кількість міст, які потрібно відвідати комівояжеру, але він повинен відвідати кожне місто лише один раз і обрати оптимальний із можливих шляхів. Структурою задачі є зважений граф без від'ємних ребр. Ціль – знаходження гамільтонового циклу з мінімальною сумою всіх ребр.

Наприклад, задано граф на рис. 1.1, оптимальним шляхом є:

$A => B => C => E => D => A$ (вага = 31)

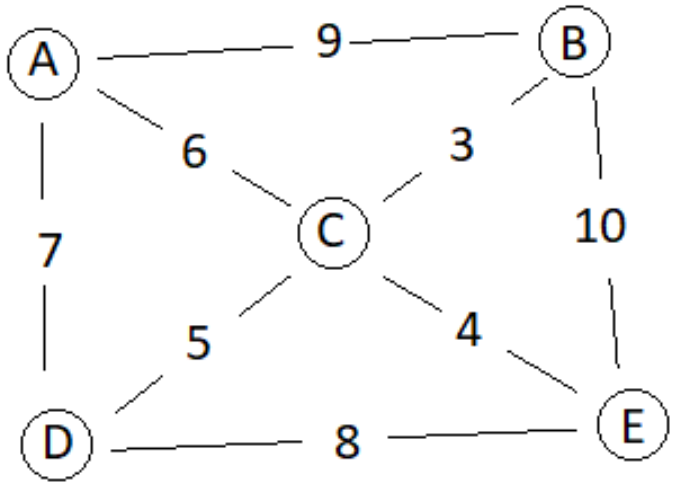

Рисунок 1.1 – зображення зваженого графу

Математичні задачі – прикладом може бути Гіпотеза Кеплера про упакування трьох-вимірних сфер однакового розміру в певну ємність. Хоч дана проблема і є вирішеною, але є велика кількість не розв’язаних математичних задач схожого типу.

Алгоритм пакування має наступний вигляд:

- Перший шар повинен бути представлений у виді гексагональної решітки.
- Кожен наступний шар потрібно розташовувати у найнижчі можливі точки між попереднім шаром.

Наближення за допомогою кривих – це процес побудови кривої, або математичної функції, яка є найбільш наближена до певного дискретного набору точок.

Деякі із типів наближень:

- Поліноміальні наближення.
- Наближення за допомогою тригонометричних функцій (таких як синус та косинус).

1.3. Відомі підходи до розв’язання задачі

Існують 2 основні категорії на які можна розбити методи глобальної оптимізації:

- Точні методи.
- Евристичні методи.

Перші методи, як правило, потребують набагато більшої кількості часу, але здатні знаходити точніші розв’язки. Евристичні ж методи є більш швидкими, але мають меншу точність.

Розглянемо деякі з точних методів.

Метод гілок і меж

Даний метод зазвичай використовують для дискретної оптимізації та комбінаторних проблем.

Мета методу – знайти таке значення x, що мінімізує значення дійсної функції $f(x)$, яку називають цільовою, серед деякої множини S, яку називають простором рішень.

Алгоритм BnB працює за такими принципами:

- Рекурсивно розбити простір рішень на малі частинки, потім мінімізувати значення функції *f(x)* на цих проміжках.
- Для покращення продуктивності даного алгоритму, BnB постійно корегує простір пошуку, якщо на якомусь проміжку неможливо знайти рішення.

Використовуючи дані принципи, можна побудувати алгоритм для потрібної оптимізаційної задачі. Саме для побудови виділяють 3 основних кроки:

- *Гілка(J)* продукує 2 або більше стани, які представляють підмножину S_j.
- *Межа(J)* обчислює значення нижньої границі пошуку для будь-якого можливого рішення в просторі рішень, визначеним S_j, таке, що *Межа(J)* $\leq f(x)$, для будь-якого значення *x* підмножини S_j.
- *Рішення(J)* визначає розв'язок для підмножини S_j.

Використовуючи дані кроки, BnB алгоритм проходить по кожній підмножині, яка була сформована на кроці *Гілка(J).* Потім йде перевірка якщо *Межа(J)* є більшою за нижню границю, яку він уже відвідав, то дана підмножина S_j відкидається.

Даний крок зазвичай фіксується глобальною змінною, у якій зберігається мінімальна границя з усіх можливих підмножин S_j.

Байесовський пошук

Базується на Байесових мережах, використовуючи, теорему Байеса:

$$P(A\,|B) = \frac{P(B \mid A) * P\ (A)}{P(B)}$$

Виділяють такі основні кроки:

1. Сформулювати всі гіпотези які можуть відбутися з об’єктом оптимізації.
2. Для кожної гіпотези визначити функцію густини ймовірностей.
3. Визначити ймовірнісну функцію знаходження об’єкта *x* (у нашому випадку мінімального рішення) у просторі *X*.
4. Об’єднавши всі функції густини ймовірностей, визначити загальну карту щільності ймовірностей. Може бути візуалізовано, як контурну карту (рис. 1.2).

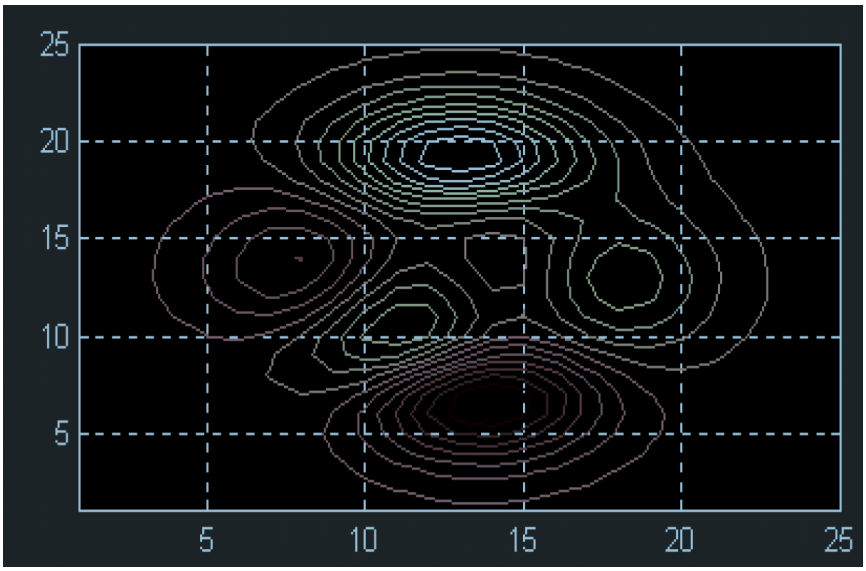

Рисунок 1.2 – приклад функції, яка має декілька областей пошуку

5. Побудувати шлях пошуку рішення, який буде починатися в точці з найбільшою ймовірністю знаходження рішення і закінчуватиметься в точці з найменшою ймовірністю.
6. Безперервно пройти по даному шляху, використовуючи певні оцінки, необхідні для знаходження рішення конкретної проблеми.

Іншими словами треба проаналізувати області з найбільшою ймовірність знаходження рішення і потім рухатись до областей з найменшою ймовірністю.

Інтегральні методи

Нехай U є деяким топологічним простором, S підмножина U і f дійсна функція багатьох змінних визначена на U.

Проблемою оптимізації є:

$c^* = \inf f(u)$, де $u \in S$

Вихідним набором оптимізації є:

$H^* = \{u \in S \mid f(u) = c^*\}$

Сам алгоритм:

Візьмемо дійсне число $c_0 > c^*$, скажімо що, $c_0 = f(u_0) + 1$, точка $u_0 \in S$, і визначимо початковий набір:

$$H_0 = \{ u \in S \mid f(u) \leq c_0 \} = H_{c_0} \cap S$$

Тоді набір H_0 є непустим, крім того, міра $H_0 > 0$. Обчислимо середнє значення та дисперсію.

$$c_1 = M(f, c_0; S) = \frac{1}{\mu(H_0)} \int_{H_0} f(u) d\mu$$

$$v_1 = V_1(f, c_0; S) = \frac{1}{\mu(H_0)} \int_{H_0} (f(u) - c_0)^2 d\mu$$

Маємо:

$$c_0 \geq c_1 \geq c^*$$

Необхідно зауважити, що якщо f є неоднорідною та неперервною на S, то.

$$c_0 > c_1 > c^*$$

Маючи c_1, можна визначити H_1:

$$H_1 = \{ x \in S \mid f(u) \leq c_1 \}$$

$$c_2 = M(f, c_1; S) = \frac{1}{\mu(H_1)} \int_{H_1} f(u) d\mu$$

$$v_2 = V_2(f, c_1; S) = \frac{1}{\mu(H_1)} \int_{H_1} (f(u) - c_1)^2 d\mu$$

Продовжуючи ітераційний процес маємо:

$$c_0 \geq c_1 \geq \cdots \geq c_k \geq c_{k+1} \geq \cdots \geq c^*$$

Можна бачити, що даний метод потребує великої кількості обчислювальних ресурсів та часу.

Розглянемо деякі евристичні методи.

Дані методи пошуку мають кращий потенціал для розв'язку задач глобальної оптимізації функцій багатьох змінних. Вони потребують меншої кількості часу, але знаходять наближений розв'язок

Генетичні алгоритми є популяційними алгоритмами, які втілюють ідею природньої еволюції Дарвіна.

Алгоритм можна поділити на такі етапи:

- Ініціалізація початкової популяції.
- Обчислення функції пристосованості.
- Відбір.
- Схрещування.
- Мутація.

Ініціалізація початкової популяцій – це задання кожного гену хромосоми випадковими можливими значеннями $l \leq x_i \leq u$.

Після цього виконується обчислення функції пристосованості для кожної хромосоми *i*-го покоління. Чим більше значення функції пристосованості, тим краще пристосована дана хромосома.

Виконується відбір між хромосомами поточної популяції, шанс потрапити у наступне покоління залежить від значення функції пристосованості.

Етап схрещування необхідний для того, щоб об’єднати інформацію між двома найближчими предками (матір’ю та батьком).

На останньому етапі формування рішення відбуваються мутації, вони мають досить малий шанс, але вплив на алгоритм мають доволі значний. Їх роль відіграє у створені додаткового простору змінних, які не з’явились на етапі ініціалізації або ж не були хибно відібрані через низьке значення функції пристосованості.

Алгоритм диференціальної еволюції

Алгоритм диференціальної еволюції є досить новим евристичним підходом для розв’язання задачі глобальної оптимізації функцій багатьох змінних. Він має 3 основні переваги відносно інших евристичних методів:

- Знаходження наближеного глобального мінімуму незалежно від параметрів ініціалізації алгоритму.
- Швидке сходження розв’язку.
- Невелика кількість параметрів для налаштування алгоритму.

Більш детально даний алгоритм буде розглянутий далі.

Симуляція відпалу

Даний алгоритм є запозиченим з природи взаємодії атомів. На виробництві його використовують в металургії. Якщо нагрівати метал до певної температури, а потім охолоджувати, то можна мінімізувати потенціальну енергію атомів в кристалічній ґратці. За допомогою таких маніпуляцій метал стає більш міцним.

Можна виділити такі основні етапи:

- Задати початкову температуру T.
- Створення початкового стану S_i.
- Випадково змінюючи попередній стан створюємо наступний стан S_{i+1}.
- Якщо S_{i+1} є кращим за S_i (зміна потенціальної енергії $\Delta E = E_2 - E_1 > 0$), то $S_i = S_{i+1}$, інакше виконуємо цю ж операцію, але з ймовірністю $eps(\frac{-\Delta E}{T})$.
- З кожною ітерацією зменшуємо температуру T.

Якщо б обирались тільки кращі рішення, то даний алгоритм був би жадібним, але процес вибору гіршого рішення з певною ймовірністю частково контролює те, що алгоритм не застряє в локальних екстремумах.

Параметр температури визначає те, на скільки ми можемо відхилитись від поточного розв’язку. В ідеальних умовах, з часом роботи алгоритму, ми будемо наближатись до необхідного рішення, але якщо температура буде великою, то шанс того, що алгоритм відхилиться від

потрібного рішення буде дуже великим. Саме через це потрібно, щоб вона постійно зменшувалась.

Досить велике значення полягає у виборі сусідніх рішень, але це залежить від конкретної задачі. Також може з’явитися необхідність перезапускати алгоритм декілька разів для більш точного знаходження глобального екстремуму, але тоді потрібно зберігати найкращі рішення.

Табу-пошук

Основний підхід полягає в тому, щоб уникати потрапляння в зациклювання за допомогою накладання заборони на деякі можливі рішення. Даний підхід є частково схожим на метод гілок та меж.

Для того, щоб метод не застрявав у локальних екстремумах існує табу-список при якому деякі рішення навіть не розглядаються. Зазвичай типи пам’яті даних списків можна поділити на 3 категорії:

- Короткочасні: Список нещодавно розглянутих рішень. Якщо з’явиться нове потенціальне рішення, то воно не буде розглянуте, поки не досягне точки вичерпання.
- Середні: Правила, які направлені на зміщення пошуку в регіони потенціальних рішень.
- Довгі: Правила, при яких пошук починає вестись в нових регіонах, якщо алгоритм знаходиться в неоптимальних регіонах.

Короткочасної пам’яті може бути достатньо для досягнення непоганих результатів для деяких задач, але якщо функція є доволі складною, то без інших типів не обійтись.

Принцип роботи:

- Генерація початкового рішення S_i.
- Генерація *n* сусідніх рішень.
- Знайти найкраще сусіднє рішення S_j.
- Якщо S_j не знаходиться в табу-списку, то $S_i = S_j$, інакше вибрати *k*-те найкраще доступне рішення.

Висновок

Евристичні методи є досить непоганими методами розв'язку задач глобальної оптимізації. Аналізуючи недоліки та переваги алгоритму диференціальної еволюції, я вирішив дослідити його у даній дипломній роботі.

2. МЕТАЕВРИСТИЧНІ АЛГОРИТМИ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ

2.1. Паралельні генетичні алгоритми(PGA)

Простий генетичний алгоритм може буде представлений у такому вигляді:

$$PGA = (\ M, C, F, o, Ps,\ Pc,\ Pm, T\ ),$$ де

M – початкова популяція, C – бітова строка кодування рішення, F – фітнес функція, Ps – ймовірність селекції, Pc – ймовірність схрещування, Pm – ймовірність мутації.

Для розв'язання NP-складних задач, широкий спектр простору збільшить довжину хромосом. Це, звичайно, збільшує складність задачі.

Паралельний генетичний алгоритм використовує 2 основні модифікації відносно базових генетичних алгоритмів. По-перше, селекція є незалежною відносно інших рішень(рис. 2.1). По-друге, кожна хромосома може покращити своє рішення з часом виконання(підняття по локальному схилу).

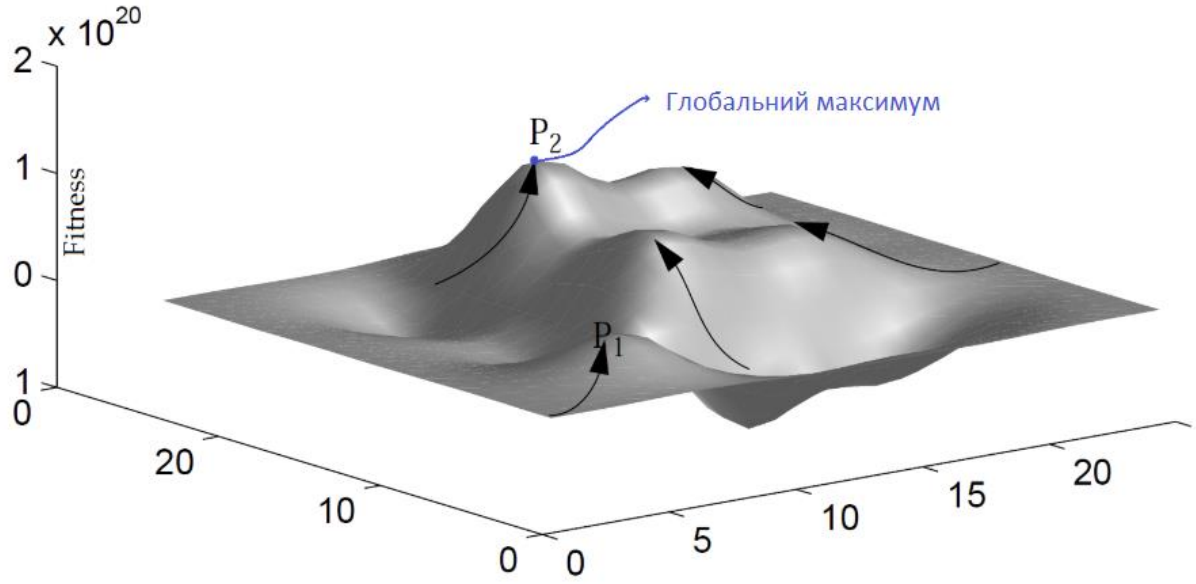

Рисунок 2.1 – випадковий пошук

PGA є повністю асинхронним, виконуючись з максимальною ефективністю з використанням MIDM. Стратегія пошуку базується на

малій кількості активних індивідуумів, коли GA використовує велику кількість пасивних членів популяції.

Схема роботи алгоритму вказана на рис. 2.2.

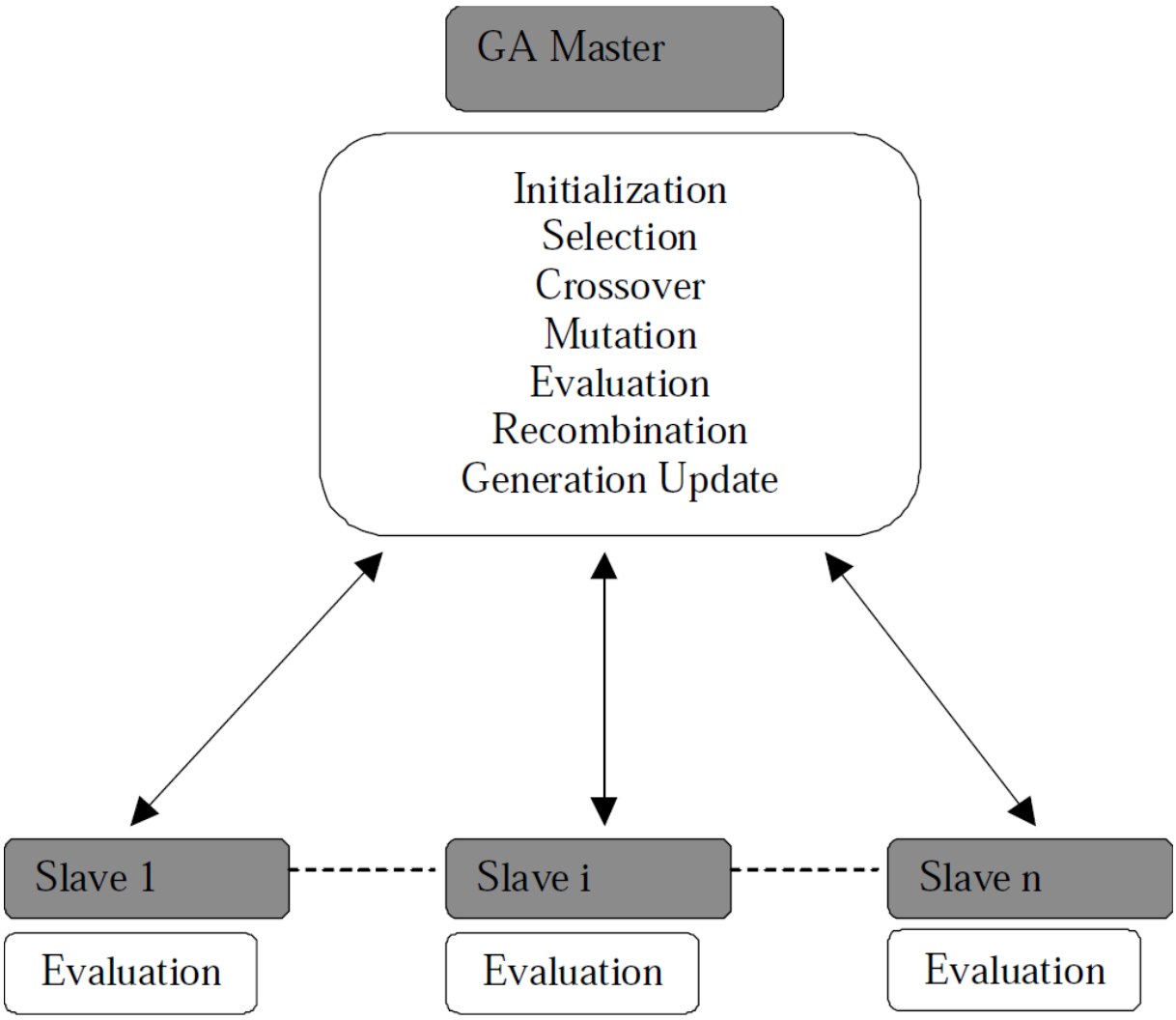

Рисунок 2.2 – схема роботи PGA

2.2. Алгоритм Grefensstett'a

Даний алгоритм використовує модифікований метод Бройдена – Флетчера – Гольдфарба - Шано задля знаходження глобальних екстремумів, який використовують для локальної оптимізації.

Псевдокод вказаний на рис. 2.3

```
1: L: Lower bound;
2: U: Upper bound of the search space.
3: n: Dimension of search space;
4: N: Population Size.
5: T_G: Total Generations/Function Evaluations (FES);
6: x ← (L + (U − L) × rand(N, n); % Initial Population of size N.
7: f(x) ← Eval_f(x); % Evaluate population set x of size N.
8: G = 1;% Initialize the generation counter.
9: while G < T_G do
10:   if rem(G, 10) == 0 then
11:     (y, f) = BFGS(x^b, f, tol)
12:   else
13:     x[G] ← Select − parents(x[G]);
14:     ý[t] ← Xovers(ý[G]);
15:     y[G] ← Mutation(ý[G]);
16:     f(y[G]) ← Evaluate(y[G]);
17:   end if
18:   if f(y[G]) < f(x[G]) then
19:     x[G]=y[G]
20:   else
21:     x[G] = x[G]
22:   end if
23:   G = G + 1;
24: end while
```

Рисунок 2.3 – псевдокод алгоритму Grefensstett'a.

2.3. Алгоритм Eshelman’a

Головна ідея алгоритму полягає у використанні ідеї елітарності та руйнівного схрещування, яке додає різноманіття популяції. Алгоритм працює з популяцією хромосом і на кожному кроці нове рішення є згенерованим, використовуючи пару рішень популяції і рекомбінуючи їх.

Даний алгоритм є нестандартною версією генетичного алгоритму. Основними відмінностями є:

- Для популяції розмірності N він гарантує, що найкращі індивідууми, які були знайдені, будуть перенесені в наступне покоління. Це гарантується тим, що селекція відбувається зі спільного списку, у якому знаходяться і батьки і діти. В базовій версії алгоритму батьківська популяція не переноситься до наступного покоління.
- Задля забезпечення завчасного зближення, 2 схожі хромосоми є розділені малою Хеммінговою відстанню.
- Під час схрещування, 2 предки обмінюються тільки половиною інформації, яку розділяє точка схрещування.
- Відсутня мутація.
- Відбувається пере ініціалізація, якщо є застій у знаходженні глобального екстремуму.

Басейн схрещування формується задаванням кожній хромосомі шансу на відтворення. Батьківська популяція є сформована використовуючи всі хромосоми даної популяції, але у випадковій послідовності.

Алгоритм Eshelman’a використовує механізм запобігання інцесту, використовуючи відстань Хеммінга, яка зазвичай рівна ¼ розмірності хромосоми. Якщо відстань між хромосомами буде менше за Хеммінгову відтань, то схрещування не буде відбуватись.

Даний метод не використовує механізм мутацій. Натомість, метод половинчастого схрещування забезпечує руйнівний рекомбінаційний механізм. Біти, які будуть схрещені обираються випадково.

Псевдокод алгоритму вказаний на рис. 2.3.

```
L: chromosome length;    p: population size
dr: divergence rate;     d: difference threshold
```

```
t = 0; d = L/4; Initialize(P(0))
repeat
    Evaluate(P(t))
    Preserve best individual from P(t − 1)
    if iCHC
        if change is detected
            P(t) = Reinitialize(P(t), dr)
    P'(t) = Selection(P(t))
    C(t) = Crossover(P'(t))
    Evaluate(C(t))
    newP(t) = Select_best(P(t), C(t))
    if newP(t) = P(t)
        decrement d
    else
        newP(t) = Reinitialize(newP(t), dr)
        d = L/4
    P(t) = newP(t)
    if RICHC
        I(t) = RndImmigate()
        Evaluate(I(t))
        P(t) = UpdatePopulation(I(t), P(t))
    if EICHC
        I(t) = EliteImmigate(E(t − 1))
        Evaluate(I(t))
        P(t) = UpdatePopulation(I(t), P(t))
    t = t + 1
until stop_condition
```

Рисунок 2.3 – псевдокод алгоритму Eshelman’a

3. РОЗРОБКА АЛГОРИТМУ ДИФЕРЕНЦІАЛЬНОЇ ЕВОЛЮЦІЇ ДЛЯ ГЛОБАЛЬНОЇ ОПТИМІЗАЦІЇ

3.1. Передумова

Глобальна оптимізація є однією з основних сфер в інженерії, статистиці та фінансах, але багато проблем мають цільову функцію, яка може буде недиференційована, перервна, нелінійна, шумна, багатовимірна або ж може мати багато локальних мінімумів, обмежень та бути стохастичною. Такі проблеми є досить важкими для їх аналітичного розв’язання, але тут можна використати алгоритм диференціальної еволюції.

Алгоритм диференціальної еволюції є досить новим евристичним підходом для розв’язання задачі глобальної оптимізації функцій багатьох змінних. Він має 3 основні переваги відносно інших евристичних методів:

- Знаходження наближеного глобального мінімуму незалежно від параметрів ініціалізації алгоритму.
- Швидке сходження розв’язку.
- Невелика кількість параметрів для налаштування алгоритму.

Основна відмінність між генетичними алгоритмами і алгоритмом диференціальної еволюції полягає в селекції та схемі мутацій, що робить DE само адаптивним. У DE всі рішення мають однакову ймовірність бути обраними батьками, незважаючи на значення функції пристосованості.

Швидкість сходження еволюційних алгоритмів є однією з найважливіших проблем, але в алгоритмі DE цей критерій було покращено.

Будучи простим, швидким, легким для використання, легко пристосованим для неперервної та дискретної оптимізації, даний метод добре себе зарекомендував для розв’язання складних проблем оптимізації.

3.2. Базовий алгоритм

DE є популяційним алгоритмом та використовує схожі операції генетичних алгоритмів:

- Схрещування.
- Мутація.
- Селекція.

Основною відмінністю для знаходження кращих рішень є те, що генетичні алгоритми покладаються на операцію схрещування, коли DE покладається на операцію мутації.

DE використовує мутації, як механізм пошуку, і операцію селекції, щоб направити пошук в перспективні регіони простору. Алгоритм використовує нерівномірне схрещування, яке може обирати дочірній вектор параметрів від одного батька частіше за іншого. Використання декількох хромосом поточної популяції дозволяє покращити пошук.

Нехай необхідно оптимізувати функцію з D дійсними параметрами. Для цього необхідно обрати розмір популяції N, мінімальна кількість хромосом повинна дорівнювати 4.

Можна виділити такі основні етапи алгоритму:

1. Ініціалізація.
2. Оцінка функції пристосованості.

3. Мутація.
4. Рекомбінація.
5. Оцінка функції пристосованості.
6. Повторювати пункти 3-5 поки критерій завершення пошуку не буде досягненим.

Ініціалізація

Основні кроки:

- Потрібно визначити верхню(U) та нижню(L) границю для кожного параметру:
  $x_j^L \leq x_{j,i,1} \leq x_j^U$
- Випадково обрати значення ініціалізації поточного вектору з проміжку $[x_j^L, x_j^U]$

Мутація в даному випадку розширює простір можливих значень. Для кожного вектору $x_{i,G}$, необхідно випадково обрати 3 вектори $x_{r1,G}$, $x_{r2,G}$, $x_{r3,G}$(саме через це популяція повинна вміщувати мінімум 4 особи) такі, що індекси *i*, *r1*, *r2* та *r3* є різними.

Тоді донорний вектор має наступний вигляд:

$v_{i,G+1} = x_{i,G} + K * (x_{r1,G} - x_{i,G}) + F * (x_{r2,G} - x_{r3,G})$, де *F* називають коефіцієнтом масштабування, а *K* – комбінаційним коефіцієнтом.

Схрещування дозволяє перенести інформацію з успішних векторів попередньої популяції.

Тимчасовий вектор $u_{i,G+1}$ повинен включати в себе елементи цільового вектору $x_{i,G}$ та елементи донорного вектору $v_{i,G+1}$:

$$u_{i,G+1} = \begin{cases} v_{i,G+1}, & if\ rand_{j,i} \leq CR\ or\ j = I_{rand} \\ x_{i,G}, & if\ rand_{j,i} > CR\ or\ j \neq I_{rand} \end{cases}, \text{ де}$$

$$i = 1, 2, \dots, N; j = 1, 2, \dots, D$$

CR – коефіцієнт схрещування $\in [\,0, 1]$.

$rand_{j,i} \sim U[0, 1], I_{rand}$ - випадкове значення від *1* до *D*.

I_{rand} також гарантує, що $u_{i,G+1} \neq x_{i,G}$.

Селекція

Усі рішення, які належать популяції мають однаковий шанс бути обраними батьками, незалежно від значення функції пристосованості.

$$x_{i,G+1} = \begin{cases} u_{i,G+1}, & if\ f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G}, & otherwise \end{cases}, \text{ де}$$

$$i = 1, 2, \dots, N$$

Мутація, схрещування та селекція продовжуються до моменту виконання критерія зупинки.

3.3. Візуалізація роботи базового алгоритму

Нехай, вхідними параметрами алгоритму є D = 2, N = 10, F = 0.5, K = 0.5 і CR = 0.1. Візуалізація буде проводитись на Ackley's function(рис. 3.1 - 3.2).

Маємо:

$$f(x_1, x_2) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{n}(x_1^2 + x_2^2)}\right) - \exp\left(\frac{1}{n}\left(\cos(2\pi x_1) + (\cos(2\pi x_2))\right)\right)$$

Необхідно знайти $x^* \epsilon [-5,5]$ такий, що $f(x^*) \leq f(x) \; \forall \; x \; \epsilon [-5,5]$ Розв'язком є $f(x^*) = 0$ при $x^* = (0,0)$

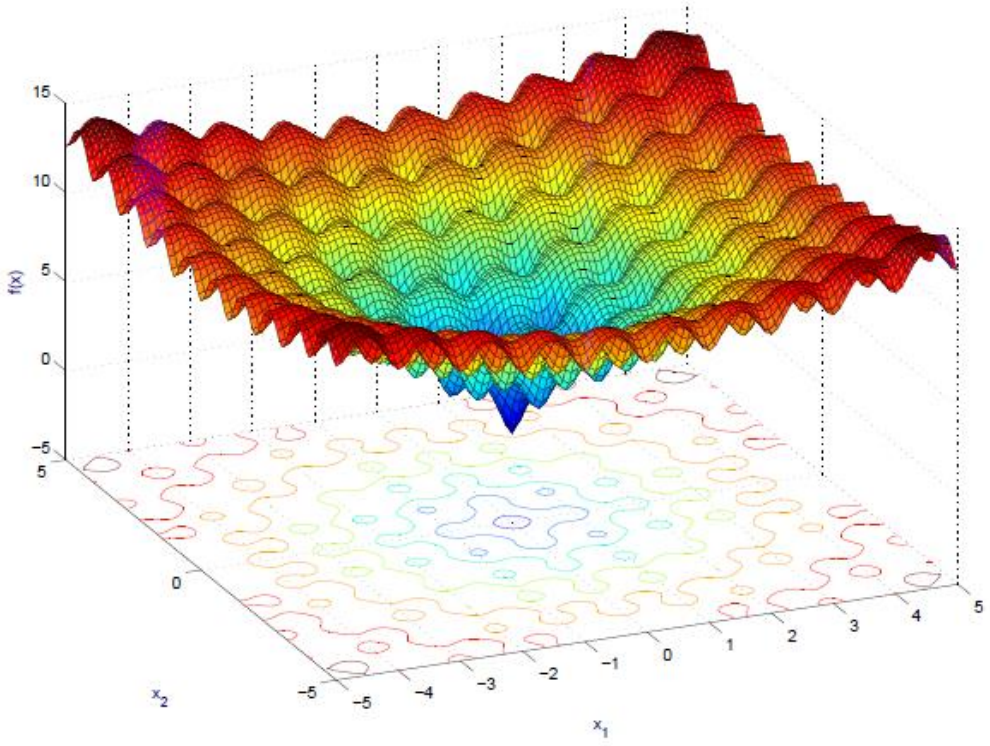

Рисунок 3.1 – зображення Ackley's function

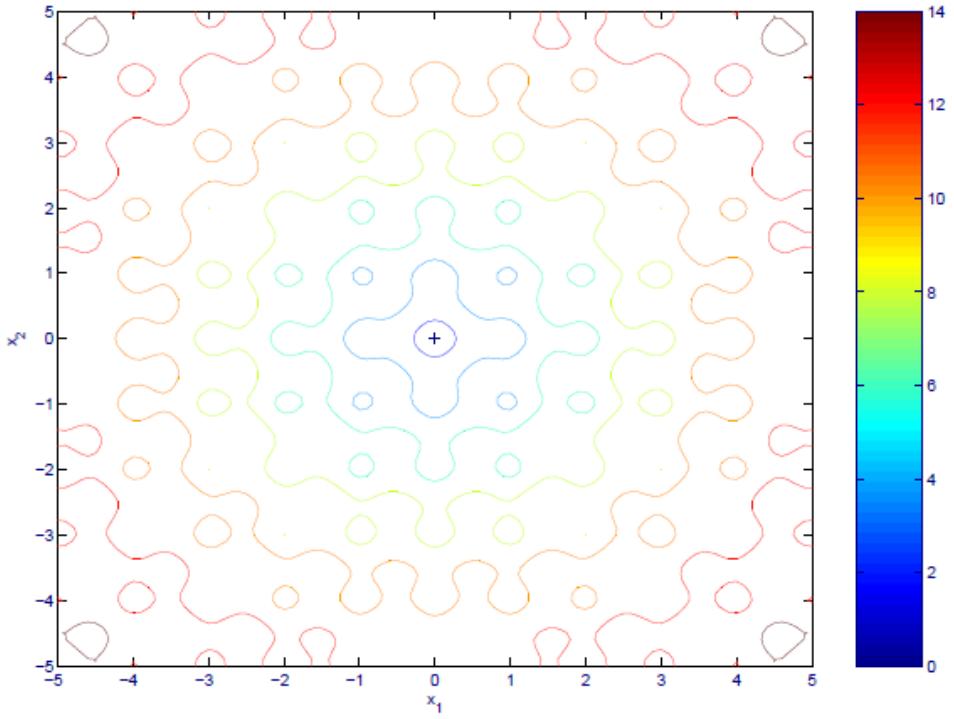

Рисунок 3.2 – розподілення значень Ackley’s function

Ініціалізація приймає наступний вигляд(рис. 3.3)

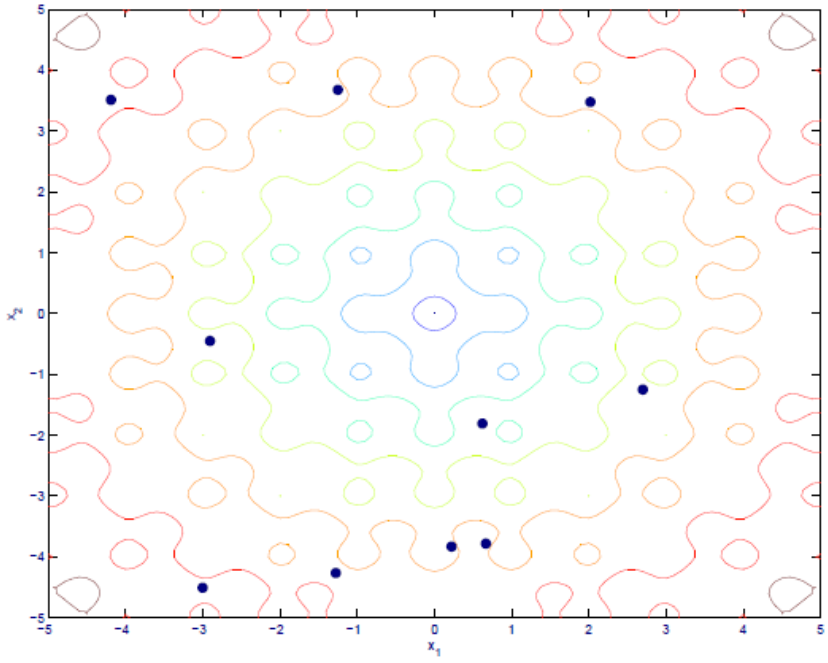

Рисунок 3.3 – ініціалізація алгоритму $N = 10$ хромосомами

Мутація приймає наступний вигляд(рис. 3.4 – 3.8)

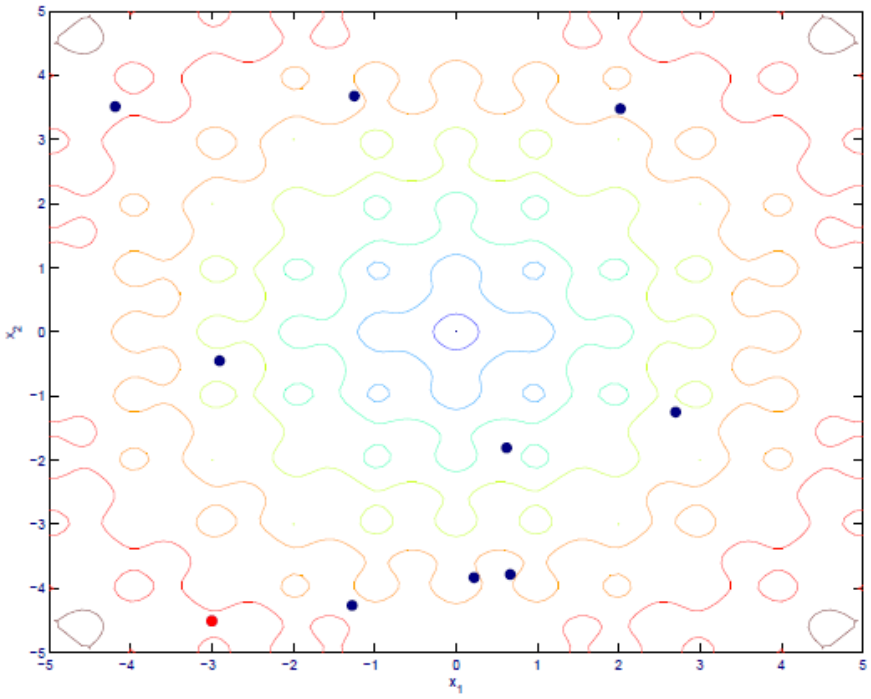

Рисунок 3.4 – вибір хромосоми, у якій буде відбуватись мутація

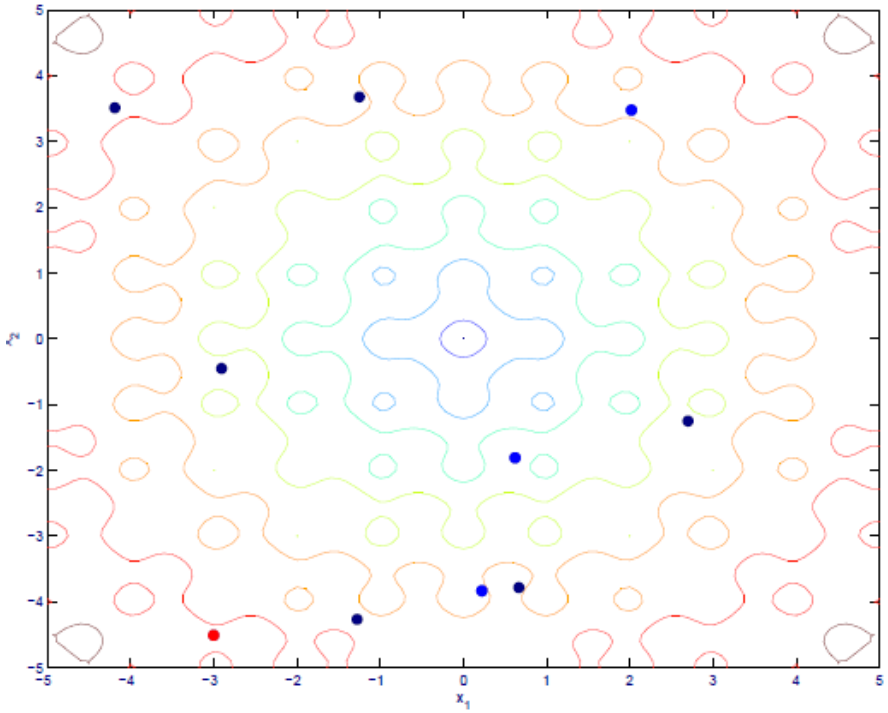

Рисунок 3.5 – вибір трьох випадкових хромосом популяції

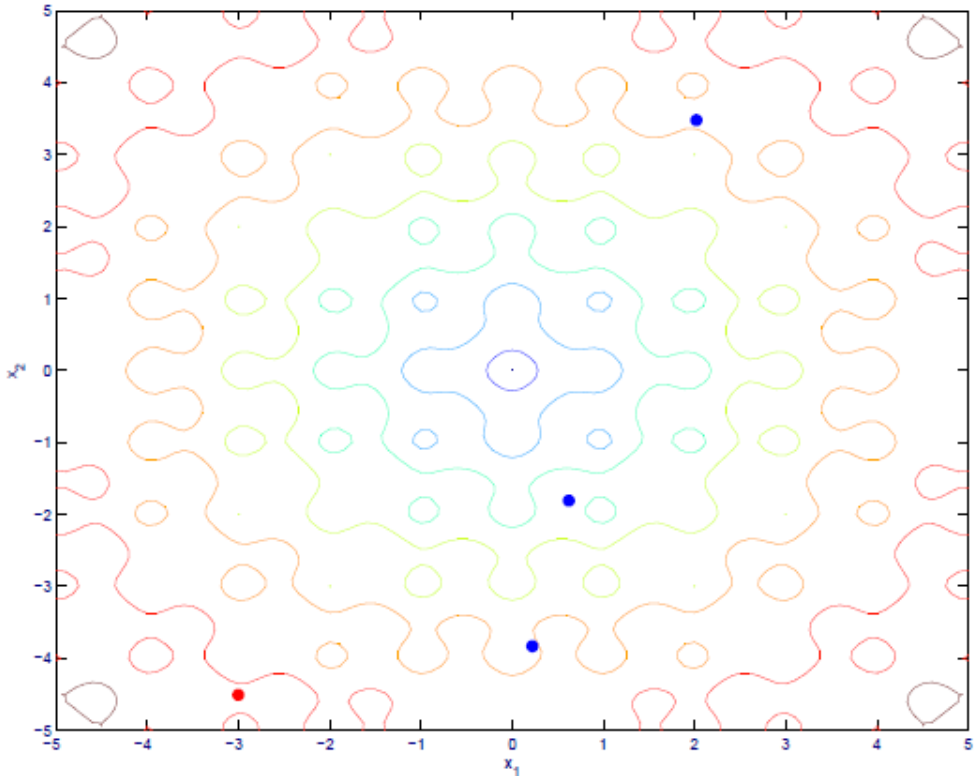

Рисунок 3.6 – елементи, які використовуються для створення тимчасової хромосоми

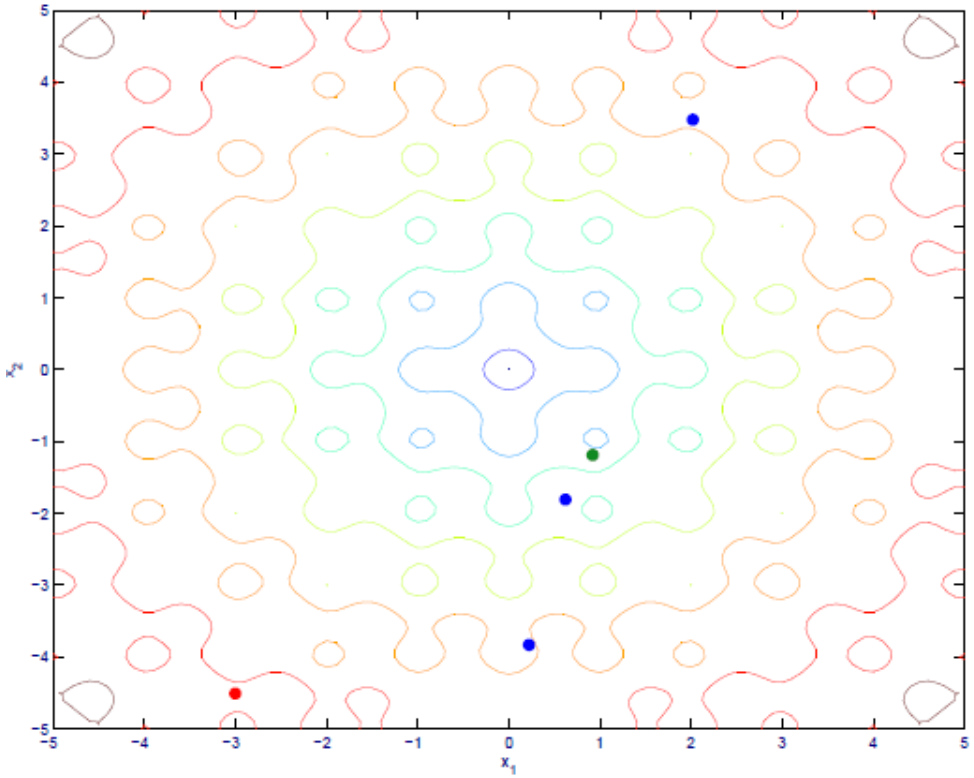

Рисунок 3.7 – генерація тимчасової хромосоми

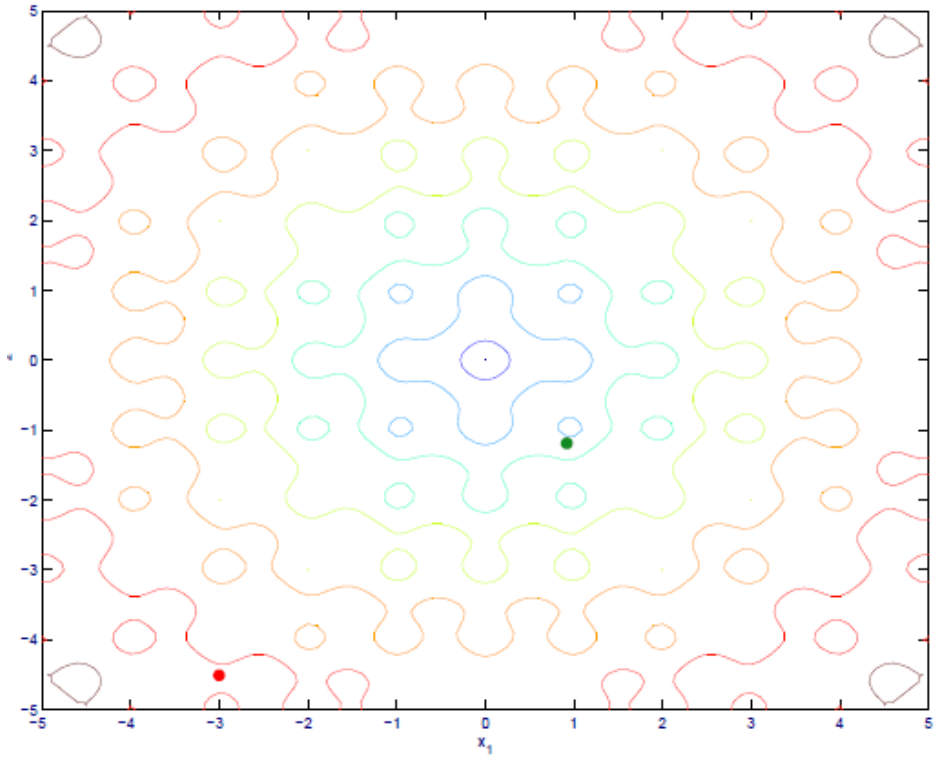

Рисунок 3.8 – вихідні елементи кроку мутації

Схрещування приймає наступний вигляд(рис 3.9)

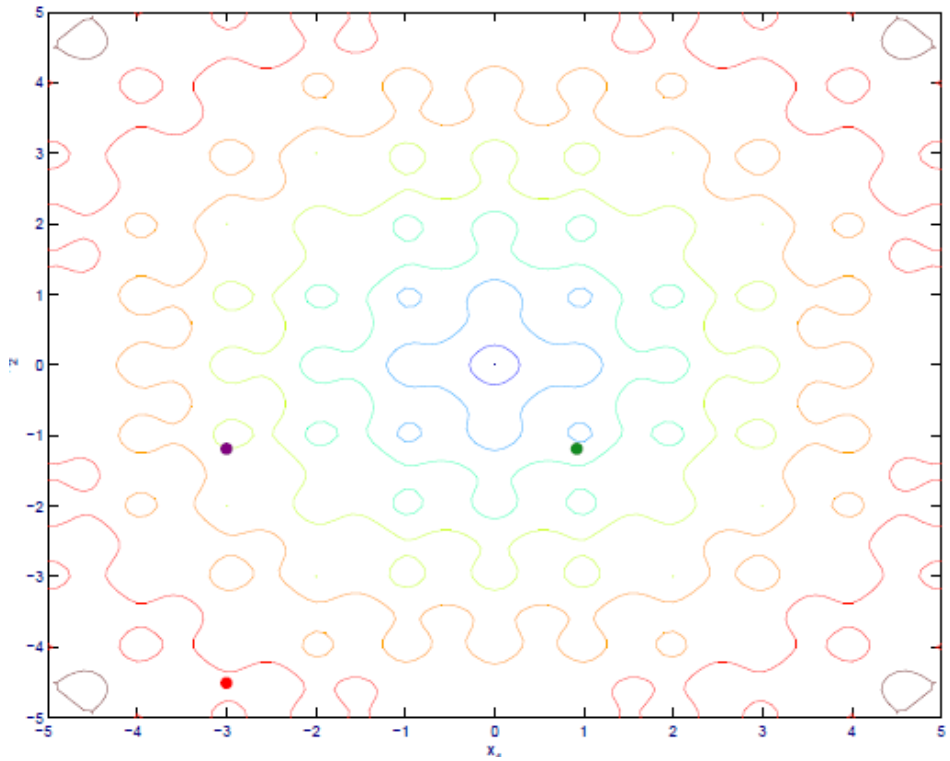

Рисунок 3.9 – схрещування тимчасової та поточної хромосоми

Селекція приймає наступний вигляд(рис 3.10 – 3.12)

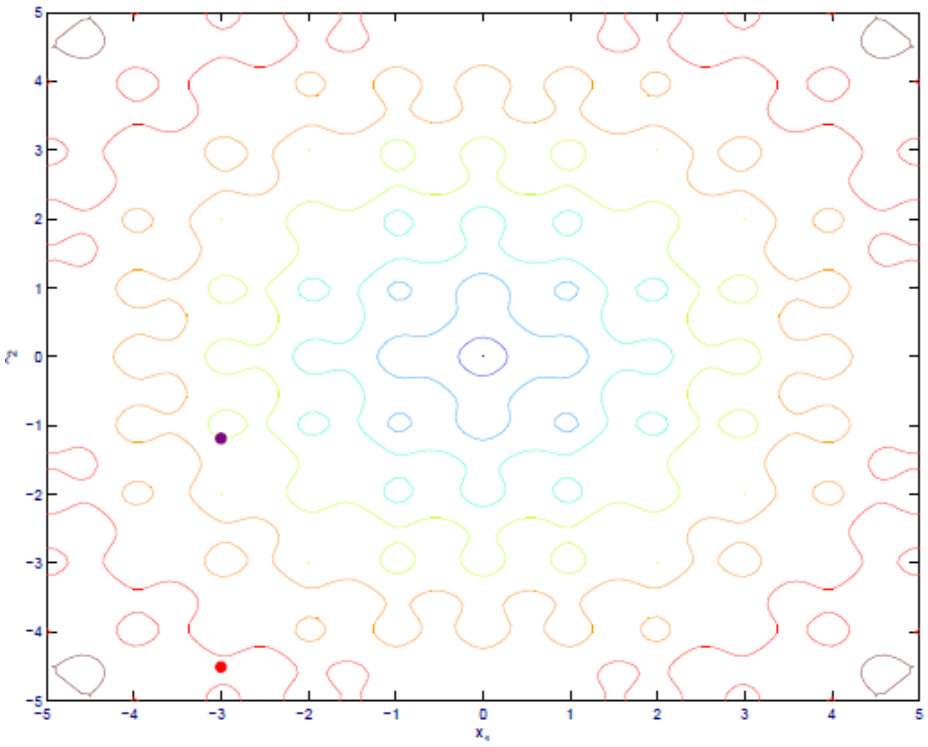

Рисунок 3.10 – поточний вектор та схрещений

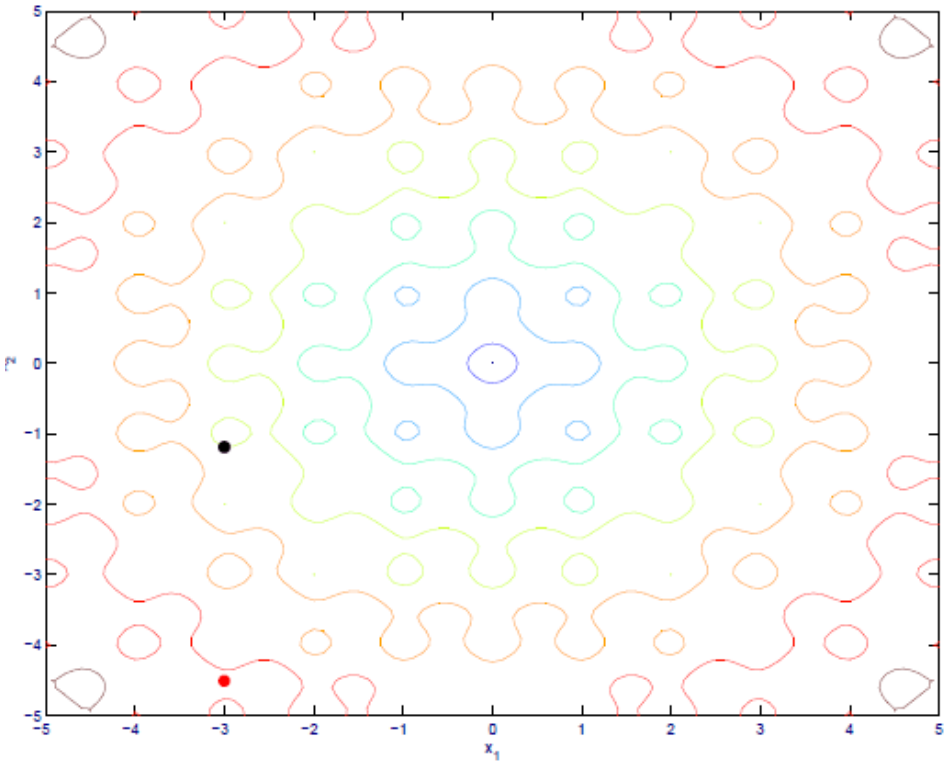

Рисунок 3.11 – схрещений вектор має більше значення функції пристосованості

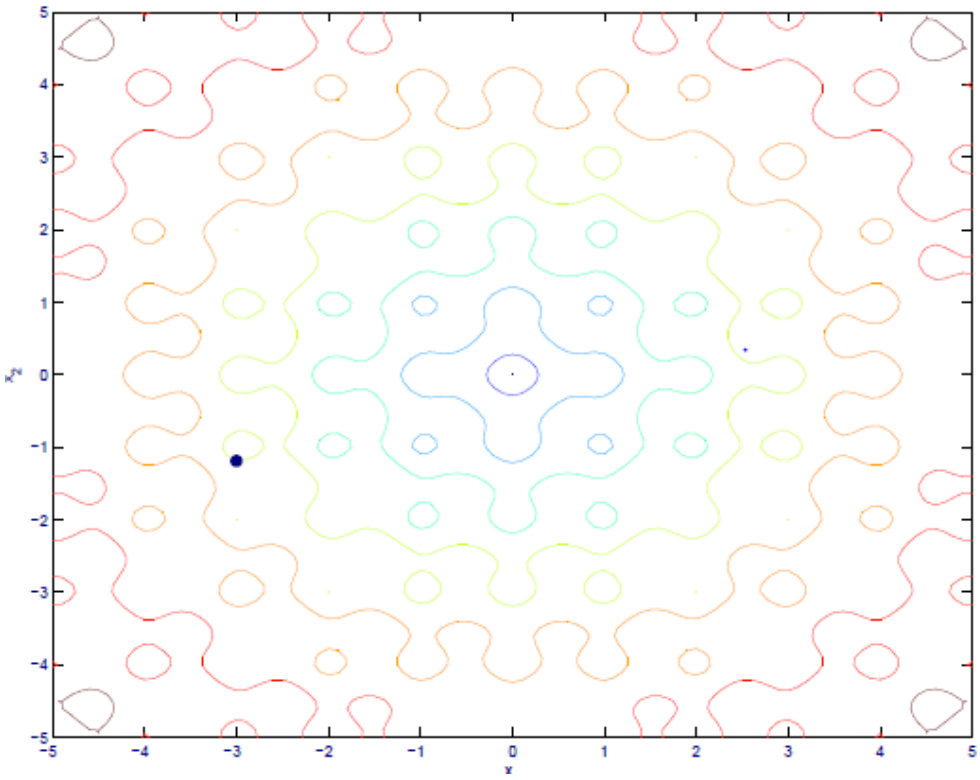

Рисунок 3.12 – приймається схрещений вектор

Після скороченого повторення даних пунктів маємо:

Мутація іншого вектору(рис 3.13)

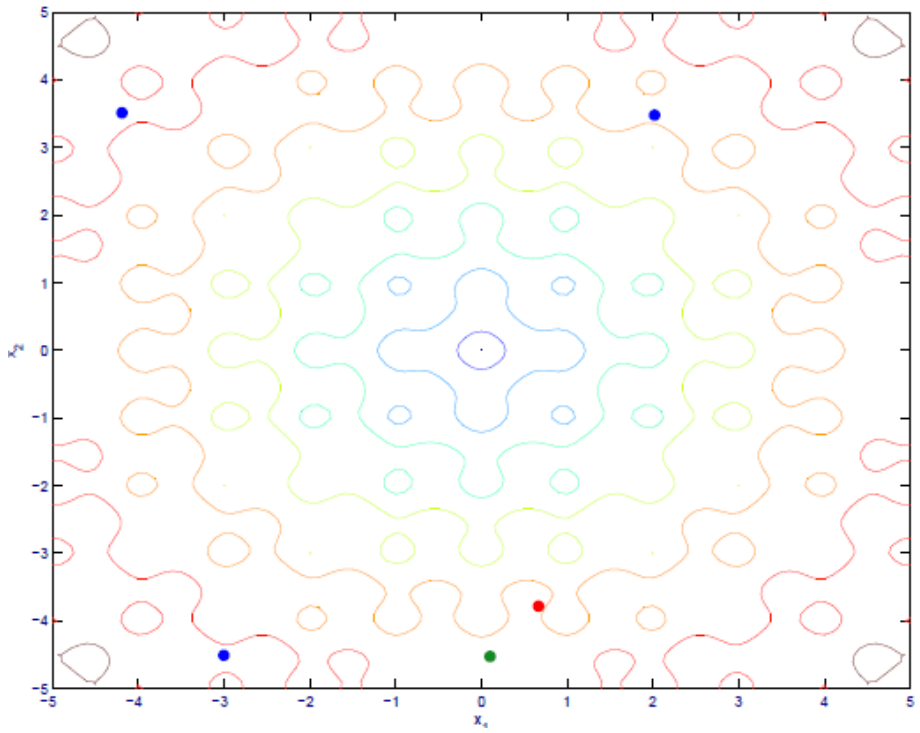

Рисунок 3.13 – приклад мутації 2-го вектору

Рекомбінація іншого вектору(рис 3.14)

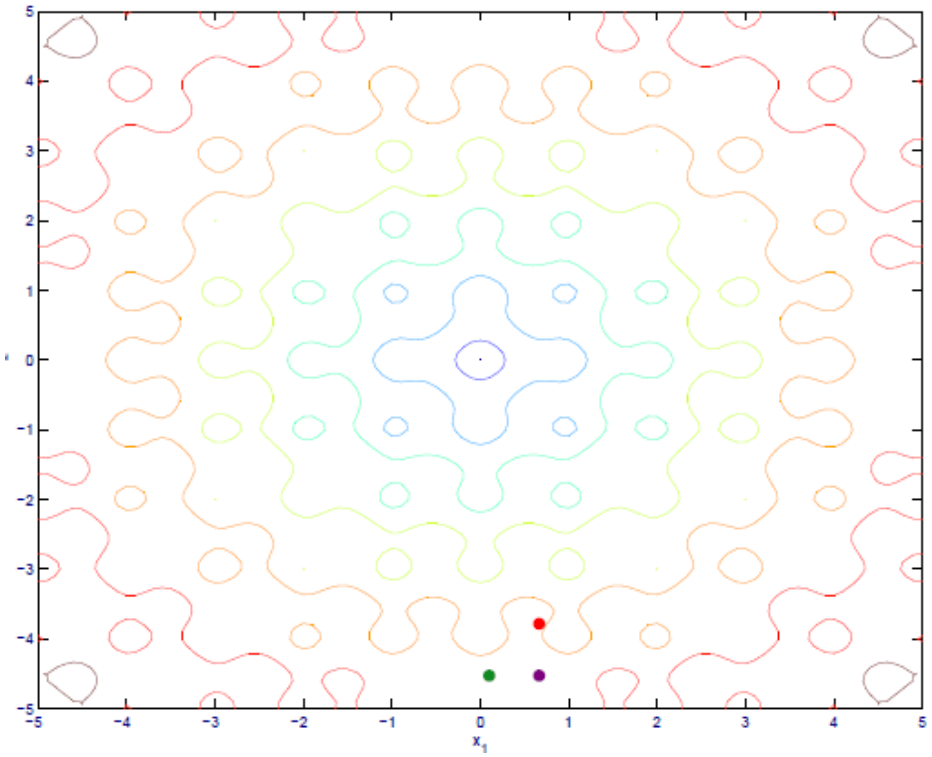

Рисунок 3.14 – приклад рекомбінації 2-го вектору

Та **селекція**(рис 3.15)

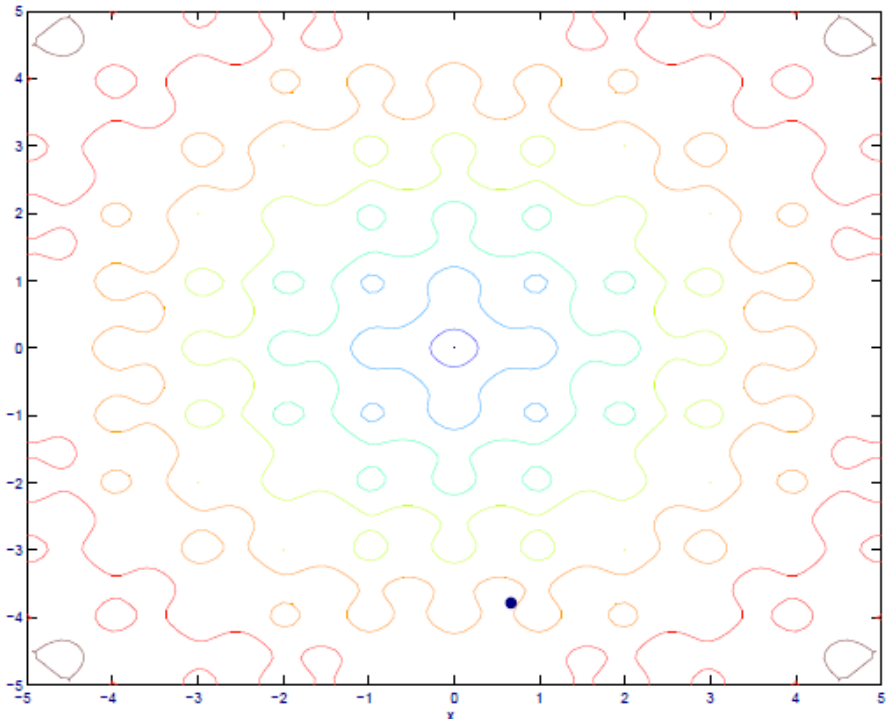

Рисунок 3.15 – приклад селекції 2-го вектору

При повторенні всіх пунктів маємо друге покоління(рис 3.16)

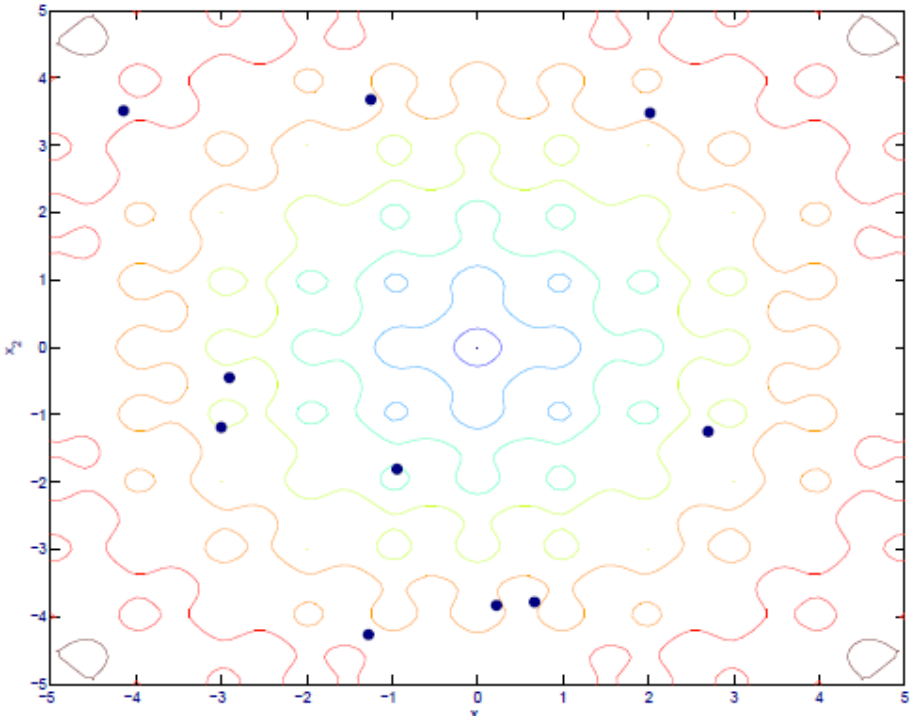

Рисунок 3.16 – 2-ге популяційне покоління

Порівнюючи з першим(рис 3.17), маємо покращення стану популяції.

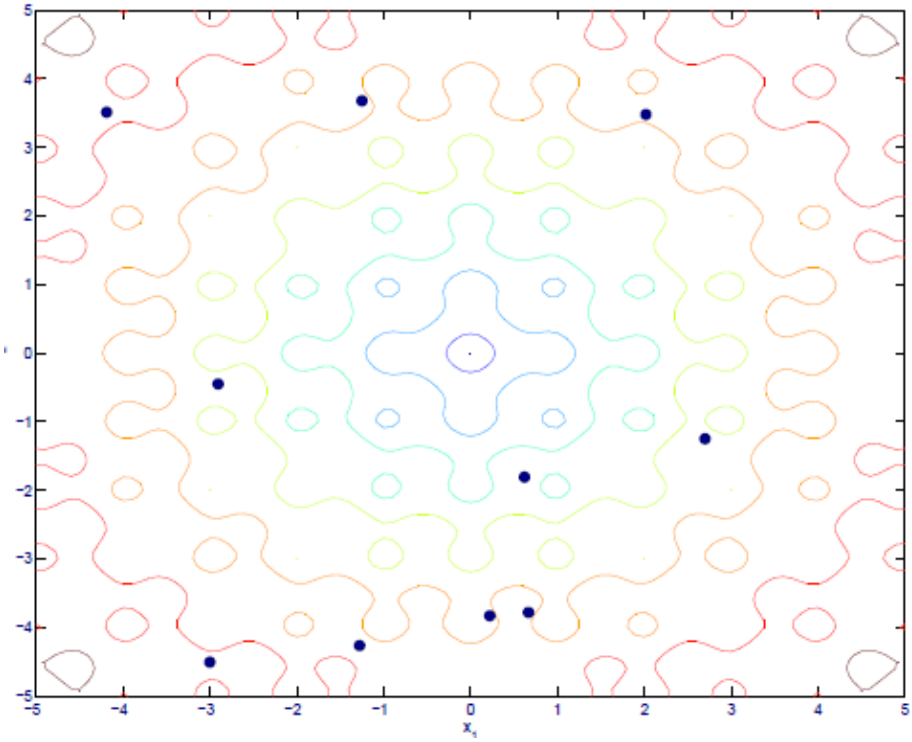

Рисунок 3.17 – перше популяційне покоління

Даний процес сходження можна представити у такому вигляді(рис 3.15)

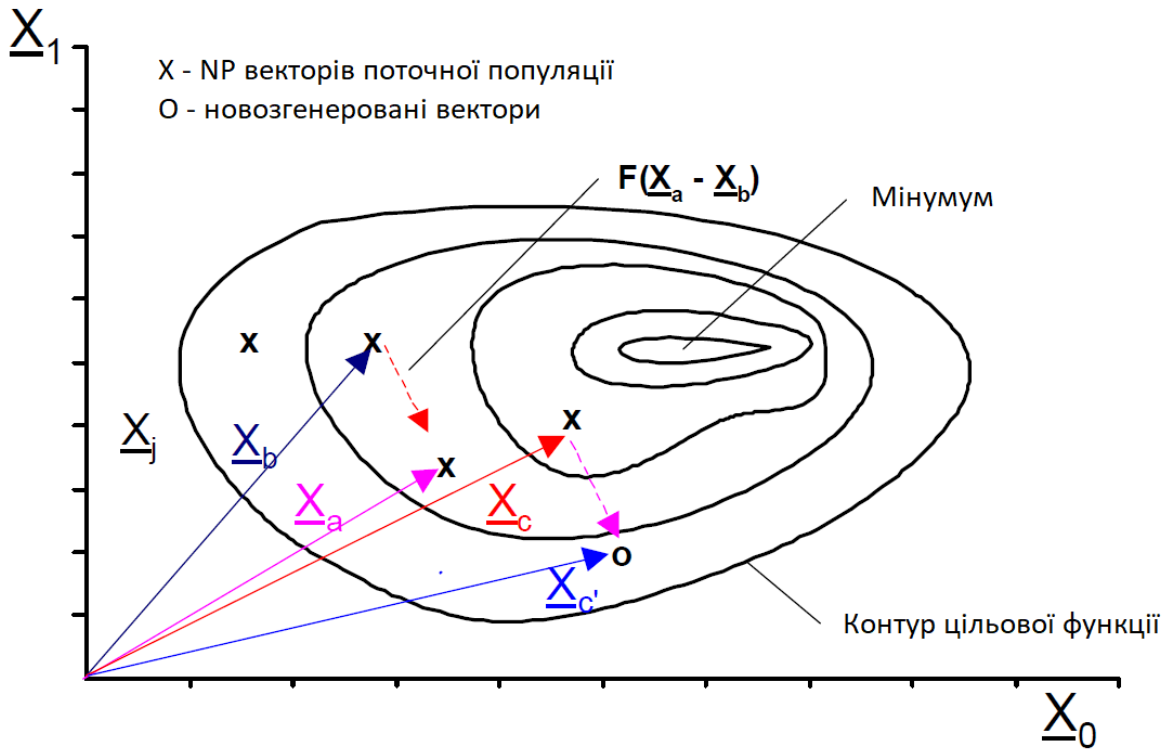

Рисунок 3.15 – зображення процесу знаходження наближеного рішення

На даному зображені можна спостерігати, як виконується оцінка поточних рішень відносно новозгенерованих. Новозгенерований вектор приймається за умови, що значення цільової функції в ньому буде меншим(або ж більшим за умови максимізації).

3.4. Модифікований алгоритм диференціальної еволюції

Основна ідея полягає в заміні формули мутації та постійній зміні налаштувань алгоритму, використовуючи архів рішень. Це дозволяє зробити алгоритм диференціальної еволюції само-адаптивним.

Схема мутацій приймає наступний вигляд:

$v_{i,G+1} = x_{i,G} + F_i * \left(x_G^{best} - x_{i,G}\right) + F_i * (x_{r2,G} - x_{r3,G})$, де

x_G^{best} – найкраще рішення G-го покоління.

Псевдокод алгоритму вказаний на рис. 3.15

Початок
Задати $\mu_{CR} = 0.5; \mu_F = 0.5; A = empty$
Створити випадкову популяцію X
for g = 1 до G
    $S_F = empty; S_{CR} = empty;$
    for i = 1 to P
        $CR_i = rand(\mu_{CR}, 0.1), F_i = rand(\mu_F, 0.1)$
        Знайти найкраще рішення в популяції x_G^{best}
        Випадково обрати $x_{r1,G}$ з популяції G та $x_{r2,G}$ з популяції $G \cup A$
$(x_{r1,G} \neq x_{r2,G} \neq x_G^{best})$
        $v_{i,G+1} = x_{i,G} + F_i * \left(x_G^{best} - x_{i,G}\right) + F_i * (x_{r2,G} - x_{r3,G})$
        Згенерувати $j_{rand} = randint(1, D)$
        for j = 1 to D
            $if\ (j = j_{rand}\ or\ rand(0,1) < CR_i)$
                Прийняти $v_{i,G+1}$
            else
                Прийняти $x_{i,G}$
        Selection$(x_{i,G}, u_{i,G})$ → додання рішення в архів A
    Випадково прибрати рішення з A, щоб $|A| \leq P$
    $\mu_{CR} = (1 - c) * \mu_{CR} + c * mean_A(S_{CR})$
    $\mu_F = (1 - c) * \mu_F + c * mean_L(S_F)$

Рисунок 3.15 – псевдокод модифікованого алгоритму диференціальної еволюції

При кожній ітерації значення CR_i та F_i є обчисленими відносно функції нормального розподілу Гауса з стандартним відхиленням 0.1, яке повинно бути відносно малим числом, так як μ_{CR} забезпечує вдалість попереднього схрещування, а F_i – попередньої мутації:

$$CR_i = rand(\mu_{CR}, 0.1), F_i = rand(\mu_F, 0.1)$$

μ_{CR} та μ_F, є ініціалізованими на початку значенням 0.5, яке постійно змінюється на кожній ітерації алгоритму:

$$\mu_{CR} = (1 - c) * \mu_{CR} + c * mean_A(S_{CR})$$

, де

$$\mu_{F_i} = (1 - c) * \mu_F + c * mean_L(S_F)$$

C – константа [0, 1].

Обчислення *mean* (середнє по Лемеру) відбувається наступним чином:

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F}$$

4. ТЕСТУВАННЯ АЛГОРИТМУ

4.1. Функції використані при тестуванні

Функції для тестування базової версії алгоритму диференціальної еволюції

F1 (Sphere function) – є випуклою, неперервною, має лише один локальний мінімум (рис. 4.1).

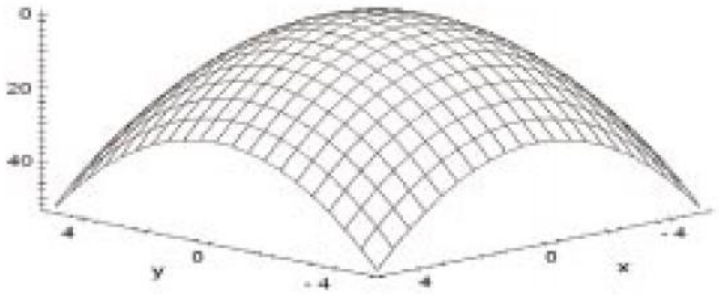

Рисунок 4.1 – зображення Sphere function

$$f(x) = \sum_{i=1}^{3} x_i^2, -5.12 \leq x_i \leq 5.12$$

F2 (Rosenbrock's function) – є неперервною, має лише один локальний мінімум. Знаходження глобального мінімуму є тривіальною задачею, але процес сходження може буде складним для деяких алгоритмів (рис. 4.2).

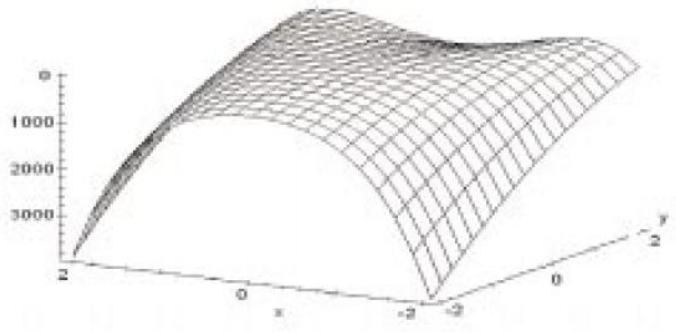

Рисунок 4.2 – зображення Rosenbrock's function

$$f(x) = 100 * (x_1^2 - x_2^2)^2 + (1 - x_1)^2 , -2.048 \leq x_i \leq 2.048$$

F3 (кусочно-задана функція) – представляє собою набір плоских регіонів, які не дають ніякої інформації про напрям руху алгоритму. Якщо певний алгоритм не може корегувати величину кроку, то можливе застрягання на плоских поверхнях (рис. 4.3).

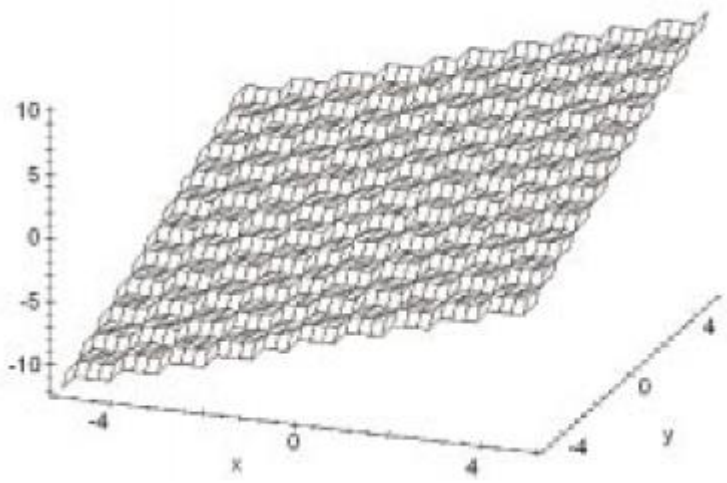

Рисунок 4.3 – зображення кусочно-заданої функції

$$f(x) = \sum_{i=1}^{5} int(x_i), -5.12 \leq x_i \leq 5.12$$

F4 (стохастична функція) – шумна унімодальна функція, яка має багато локальних екстремумів. Кожне наступне значення в певній точці є іншим. Є складною для більшості алгоритмів (рис. 4.4)

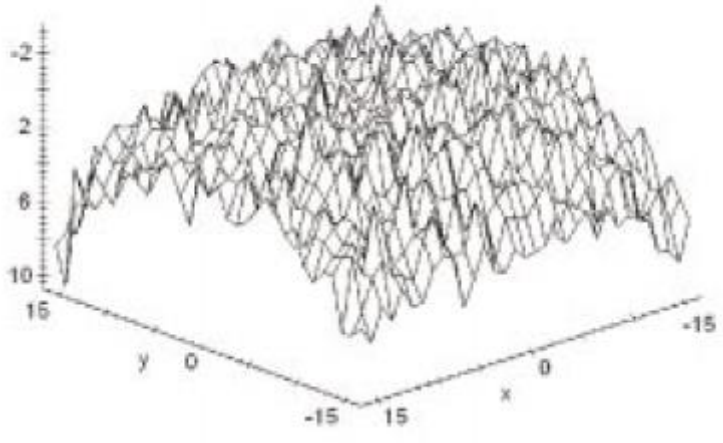

Рисунок 4.4 – зображення стохастичної функції

$$f(x) = \sum_{i=1}^{30} ix_i^4 + Gauss(0,1), -1.28 \leq x_i \leq 1.28$$

F5 (Shekel's foxholes function) – включає велику кількістю локальних екстремумів. Більшість алгоритмів застрягають в першому локальному екстремумі(рис 4.5)

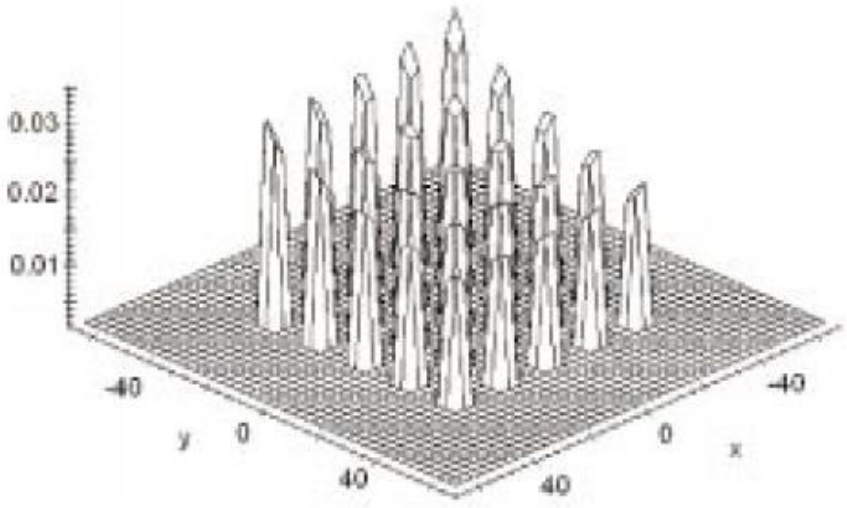

Рисунок 4.5 – зображення Shekel's foxholes function

$$f(x) = int(0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}, -65.536 \le x_i \le 65.536$$

Також для оцінки модифікованої версії алгоритму відносно базової було використано такі функції(рис 4.6).

Функція	Межа пошуку
$f_1(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^D$
$f_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]^D$
$f_3(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	$[-100, 100]^D$
$f_4(x) = \max_i \{\lvert x_i \rvert\}$	$[-100, 100]^D$
$f_5(x) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$[-30, 30]^D$
$f_6(x) = \sum_{i=1}^{D} \lfloor x_i + 0.5 \rfloor^2$	$[-100, 100]^D$
$f_7(x) = \sum_{i=1}^{D} i x_i^4 + rand\,[0, 1)$	$[-1.28, 1.28]^D$
$f_8(x) = \sum_{i=1}^{D} -x_i \sin\sqrt{\lvert x_i \rvert} + D \cdot 418.98288727243369$	$[-500, 500]^D$
$f_9(x) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]^D$
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e$	$[-32, 32]^D$
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^D$
$f_{12}(x) = \frac{\pi}{D}\{10\sin^2(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_D - 1)^2\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$ where $y_i = 1 + \frac{1}{4}(x_i + 1)$ and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$	$[-50, 50]^D$
$f_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2\left[1 + \sin^2(3\pi x_{i+1})\right] + (x_D - 1)^2\left[1 + \sin^2(2\pi x_D)\right]\} + \sum_{i=1}^{D} u(x_i, 5, 100, 4)$	$[-50, 50]^D$

Рисунок 4.6 – функції для тестування модифікованої версії алгоритму диференціальної еволюції

4.2. Дослідження впливу налаштувань алгоритму

Для базового алгоритму тестування виконувалось 50 разів.

Результати вказано на рис. 4.7.

F	F1 (N = 10, K = 0.5, CR = 0.8)	F2 (N = 10, K = 0.5, CR = 0.8)	F3 (N = 20, K = 0.5, CR = 0.8)	F4 (N = 20, K = 0.5, CR = 0.8)	F5 (N = 10, K = 0.5, CR = 0.8)
0.2	232	INF	158	1090	INF
0.4	175	INF	150	1218	INF
0.6	201	INF	145	1774	INF
0.8	268	815	135	6521	INF
1	352	760	125	INF	735
1.5	683	1254	117	INF	1622
2	1175	1762	95	INF	3183

Рисунок 4.7 - результат тестування параметра *F* базового алгоритму (INF означає, що близький розв’язок не було знайдено за 10000 поколінь)

Можна спостерігати, що параметр *F* має великий вплив на сходження базового алгоритму диференціальної еволюції та має бути налаштованим безпосередньо для кожної функції.

Для модифікованої версії непотрібно проводити дослідження впливу налаштувань, так як алгоритм є само-адаптивним.

4.3. Порівняння базової версії алгоритму відносно інших метаевристичних методів

Кожен алгоритм був протестований 50 разів. Для DE значення *F* обиралось випадково з проміжку [-2; 2].

Результати вказані на рис. 4.8.

Алгоритм	F1	F2	F3	F4	F5
PGA($\lambda = 4$)	1170	1235	3481	3194	1256
PGA($\lambda = 8$)	1526	1671	3634	5243	2076
Grefensstette	2210	14229	2259	3070	4334
Eshelman	1538	9477	1740	4137	3004
DE	274	654	132	2184	1321

Рисунок 4.8 – середня кількість популяцій

Розмірність функцій та межі пошуку рішення були вказані в розділі 4.1.

Базова версія алгоритму значно випереджає вказані метаевристичні методи за кількістю необхідних популяцій для знаходження наближеного рішення. Якщо для кожної функції окремо налаштувати значення коефіцієнта масштабування, то можна отримати кращі результати.

4.4. Порівняння базової та модифікованої версії алгоритму

Для кожного алгоритму проводилось 50 незалежних запусків.

Результати зображені на рис. 4.9 – 4.10.

Функція	Генерацій	MDE Mean (Std. Dev)	DE Mean (Std. Dev)
F1	2000	1.3E–54 (9.2E–54)	9.8E–14 (8.4E–14)
F2	3000	3.9E–22 (2.7E–21)	1.6E–09 (1.1E–09)
F3	8000	6.0E–87 (1.9E–86)	6.6E–11 (8.8E–11)
F4	15000	4.3E–66 (1.2E–65)	4.2E–01 (1.1E+00)
F5	6000	3.2E–01 (1.1E+00)	2.1E+00 (1.5E+00)
	20000	3.2E–01 (1.1E+00)	8.0E–02 (5.6E–01)
F6	100	5.6E+00 (1.6E+00)	4.7E+03 (1.1E+03)
	1500	0.0E+00 (0.0E+00)	0.0E+00 (0.0E+00)

F7	6000	6.8E–04 (2.5E–04)	4.7E–03 (1.2E–03)
F8	1000	7.1E+00 (2.8E+01)	5.9E+03 (1.1E+03)
	9000	7.1E+00 (2.8E+01)	5.7E+01 (7.6E+01)
F9	3000	1.4E–04 (6.5E–05)	1.8E+02 (1.3E+01)
	9000	0.0E+00 (0.0E+00)	7.1E+01 (2.1E+01)
F10	500	3.0E–09 (2.2E–09)	1.1E–01 (3.9E–02)
	3000	4.4E–15 (0.0E+00)	9.7E–11 (5.0E–11)
F11	500	2.0E–04 (1.4E–03)	2.0E–01 (1.1E–01)
	3000	2.0E–04 (1.4E–03)	0.0E+00 (0.0E+00)
F12	500	3.8E–16 (8.3E–16)	1.2E–02 (1.0E–02)
	3000	1.6E–32 (5.5E–48)	1.1E–14 (1.0E–14)

Рисунок 4.9 - тестування модифікованої версії алгоритму диференціальної еволюції (D = 30)

Функція	Генерацій	MDE Mean (Std. Dev)	DE Mean (Std. Dev)
F1	2000	5.4E–67 (1.6E–66)	3.5E+01 (9.6E+00)
F2	3000	9.2E–51 (2.2E–50)	2.3E+00 (5.6E–01)
F3	8000	2.2E–37 (2.5E–37)	2.1E+05 (3.1E+04)
F4	15000	3.2E–71 (8.3E–71)	9.3E+01 (2.8E+00)
F5	6000	4.0E–01 (1.2E+00)	9.5E+01 (1.4E+01)

F6	100	1.2E+02 (1.3E+01)	1.8E+05 (1.8E+04)
	1500	0.0E+00 (0.0E+00)	3.2E+02 (6.3E+01)
F7	6000	7.8E–04 (1.4E–04)	2.9E–02 (5.7E–03)
F8	1000	8.6E+03 (4.2E+02)	3.2E+04 (4.7E+02)
	9000	1.1E–10 (0.0E+00)	3.0E+04 (9.0E+02)
F9	3000	2.0E–01 (3.7E–02)	8.6E+02 (2.2E+01)
	9000	0.0E+00 (0.0E+00)	8.1E+02 (1.8E+01)
F10	500	4.2E–07 (1.2E–07)	1.5E+01 (5.8E–01)
	3000	8.0E–15 (0.0E+00)	1.3E–01 (2.4E–02)
F11	500	1.5E–04 (1.0E–03)	2.7E+02 (4.4E+01)
	3000	1.5E–04 (1.0E–03)	2.0E–01 (5.8E–02)
F12	500	2.8E–13 (9.8E–13)	1.8E+09 (5.1E+08)
	3000	4.7E–33 (6.8E–49)	1.7E+00 (1.5E+00)
F13	500	5.8E–12 (5.5E–12)	2.4E+09 (1.1E+09)
	3000	1.4E–32 (1.1E–47)	5.1E+00 (3.2E+00)

Рисунок 4.10 - тестування модифікованої версії алгоритму диференціальної еволюції (D = 100)

4.5. Аналіз результатів

Для базової версії алгоритму значний вплив має значення параметра *F*. Він повинен бути налаштованим для кожної оптимізаційної функції окремо.

Середнє значення та середньоквадратичне відхилення модифікованої версії алгоритму диференціальної еволюції є набагато меншим порівнюючи з базовою версією алгоритму.

Було проведено дослідження впливу параметру

Можна бачити, що MDE є в рази кращим за базову версію DE. Це пов’язано з тим, що MDE може адаптувати свої налаштування для кожної вхідної функції та зміною формули мутації, яка пришвидшує сходження алгоритму, використовуючи найкращий розв’язок популяції.

ВИСНОВКИ

Під час виконання роботи було проаналізовано літературні джерела, досліджено метаевристичні алгоритми для розв'язання задачі глобальної оптимізації функції багатьох змінних, було розроблено алгоритм диференціальної еволюції та його модифіковану версію, порівняно базову версію алгоритму з іншими метаевристичними методами.

Було визначено, що алгоритм диференціальної еволюції є сильним інструментом глобальної оптимізації, порівнюючи відносно інших оптимізаційних методів представлених у даній дипломній роботі.

Розроблена модифікація значно випереджає базову версію алгоритму по точності знаходження рішень, швидкості сходження функції багатьох змінних та має більш якісні показники середньоквадратичного відхилення.

Було досліджено вплив налаштувань для базової версії алгоритму. Виявилося, що параметр *F*(коефіцієнт масштабування) має значний вплив на швидкість сходження алгоритму. Необхідне безпосереднє налаштування даного параметру для різних оптимізаційних функцій, щоб забезпечити швидку роботу алгоритму.

Параметри ініціалізації модифікованої версії алгоритму можуть приймати будь-які значення, так як алгоритм є само адаптованим, але для ініціалізації було обрано такі значення: $F_i = 0.5,\ CR = 0,$ де CR – коефіцієнт схрещування. З часом роботи вони будуть оптимізовані, використовуючи стандартне відхилення на 0.1.

Тестування проводилось на відомих функціях оцінки евристичних методів. Само адаптивність модифікованої версії алгоритму та змінена формула мутацій має значний вплив на продуктивність роботи алгоритму. Як мінімум 10 із 13 функцій оцінювання представлених в даній роботі відобразили кращий результат відносно базової версії алгоритму.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

[1] K. Price, New Ideas in Optimization, McGraw-Hill Publishing Company, pp. 77-106, 1999.

[2] R. Storn, "Differential Evolution, A Simple and Efficient Heuristic Strategy for Global Optimization over Continuous Spaces", Journal of Global.

[3] M. Strens and A. Moore, “Policy Search using Paired Comparisons”, Journal of Machine Learning Research, Vol. 3, pp. 921-950, 2002.

[4] H.A. Abbass, Ruhul Sarker, and Charles Newton. “PDE: A pareto-frontier differential evolution approach for multi-objective optimization problems”. Proceedings, 2001.